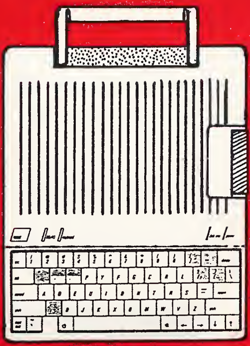
APPLE II SERIES

StripWare™

JOHN WILEY & SONS

THE BASIC APPLE //c



Includes 3 programs:

- Jumble Puzzle
- Paint Brush
- Bar Graph

Package is not returnable if plastic wrap is cut or removed.

Softstrip™
COMPUTER READABLE PRINT

Suggested Retail Price

$6.98

- Create low resolution graphics.
- Make your own word jumbles.
- Design bar charts to help you analyze important data.

CAUZIN SYSTEMS, INC.
835 SOUTH MAIN STREET, WATERBURY, CT 06706

OTHER TITLES AVAILABLE IN STRIPWARE™

FOR APPLE

Art and Graphics on the Apple II/IIe
Essential Small Business Planning
Home Education, Vol. 1
Arcade Games, Vol. 1
Disk Doctor
Computer Puzzles
Balance Sheet

FOR IBM

PC Graphics
Balance Sheet
Classic Games
Eliza
Celestia
Personal Calendar
Home Applications

Softstrip
COMPUTER READABLE PRINT

- Create low resolution graphics.
- Make your own word jumbles.
- Design bar charts to help you analyze important data.

CAUZIN SYSTEMS, INC.
835 South Main Street
Waterbury, Connecticut 06706
(203) 573-0150

8501002

Welcome to the Cauzin Family

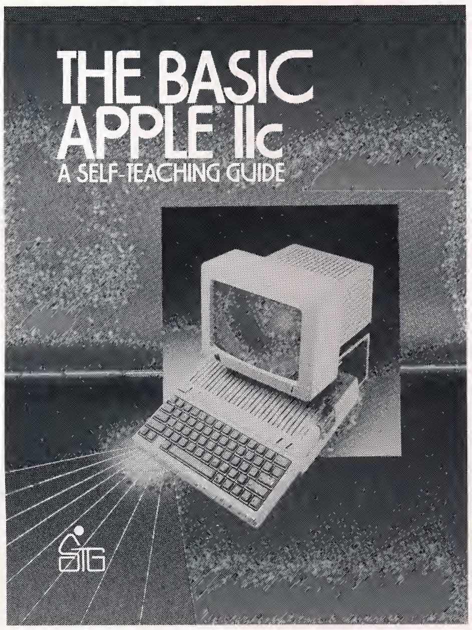
Welcome to the world of StripWare™ data strips. You'll find a new generation of computer uses unlike any you've experienced before.

You now own a portion of a revolutionary technological advancement that bridges the gap between print and electronic media more quickly and easily than ever before.

We at Cauzin Systems, Inc. hope you'll enjoy this software package, and believe you'll find data strips to be very practical and useful.

Your comments and ideas on this booklet or any additional applications you'd like to suggest would be greatly appreciated. Please send your comments or remarks to:

Customer Service
CAUZIN SYSTEMS, INC.
835 South Main Street
Waterbury, Connecticut 06706



by Gary Cornell and William Abikoff

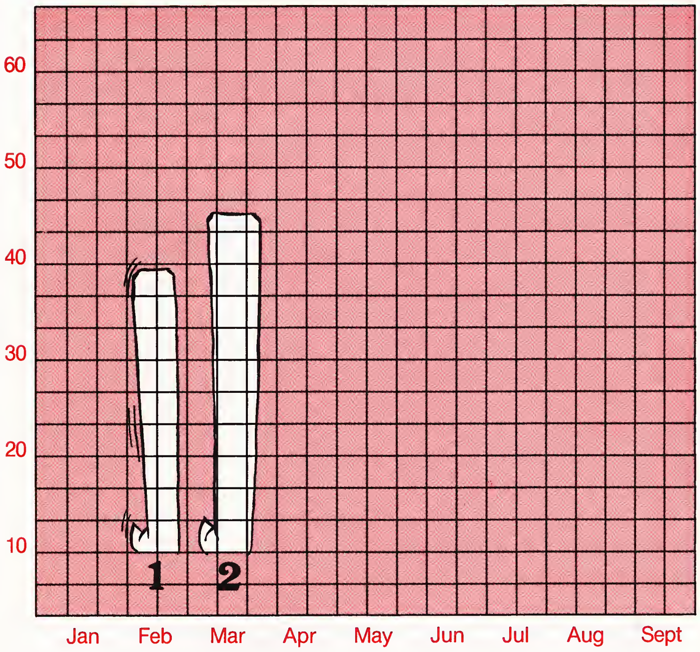
In order to master the //c rather than just use some of its features, you should learn to program it. By programming in Applesoft BASIC, the "flavor" of BASIC built into Apple // computers, you can do just that. You can't learn programming by sitting in an easy chair and reading a book (even this one). You have to sit down at the keyboard and work through the book.

This booklet will introduce you to programming in Applesoft. It will do so with graphics that add zip to reports, paint interesting designs in lo-res, and prepare word puzzles to tease the mind.

These programs are from a book of the same name, published by John Wiley & Sons, Inc., which contains over 100 similar programs that will help you learn or sharpen your BASIC programming skills.

Gary Cornell is a professor at the University of Connecticut who specializes in computer literacy.
William Abikoff, also a professor at the University of Connecticut, teaches BASIC programming to prospective elementary teachers.

A Bar Graph Program



A bar graph is a picture showing a comparison between things by piling up blocks. The piles show the relative sizes of the things. For example, if you have seventy-five books and your friend has twenty-five, you could compare these numbers by making piles of blocks. Your Apple //c and the other Apple II's can show these comparisons by making the piles from lo-res graphics blocks. The comparison is very attractive when the piles have different colors. Bar graphs are usually labelled across the bottom and have a scale along the left edge showing the size of the piles. The left edge and the bottom are called the *axes*. The horizontal edge is usually called the *x-axis* and the vertical edge is the *y-axis*.

In our bar graph, the colored bars (and labels) will stand for three different months. We'll also have the x– and y–axes in white along the bottom and left edges, respectively.

The program on this Softstrip™ data strip is for any Apple II computer. It works fine under DOS 3.3 and ProDOS.

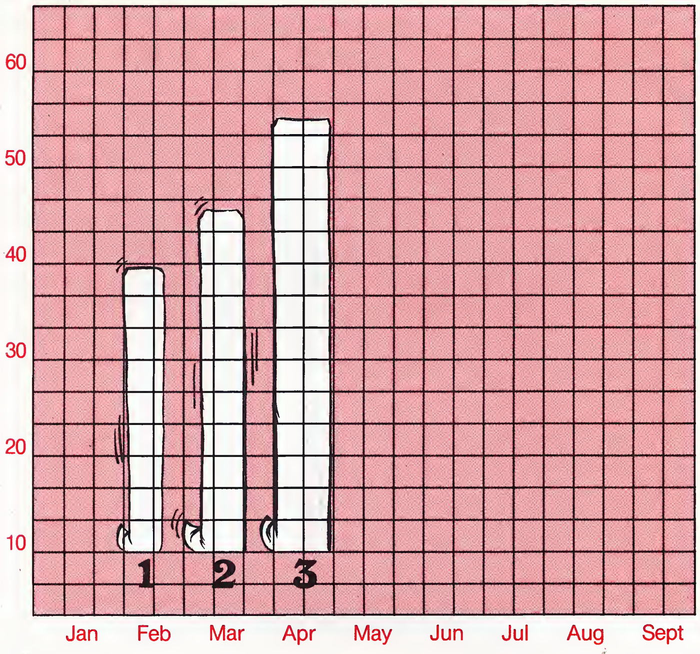
The only thing that is tricky about a program to draw bar graphs is that the bars start from the bottom of the screen and go up. Each bar represents some number of things that happened in that month. You could have seventeen units in the first month (a unit could be the sale of a car or millions in sales for a large company or even the number of times you went shopping). The bar starts at row 38, so a bar up to row 38 represents one unit. The bar has to go up sixteen more units until it ends at row 22. Since the lo-res has 40 rows, we will allow up to thirty-nine units. If you had, say, seventy-six units in a month, then each row would have to stand for two units. It wouldn't be hard to have the //c calculate a scale based on the information you gave it. Since this program is going to be fairly long as it is, we won't do scaling here.

Here is an outline of the program:

1. Get the data for the various months.
2. Display the axes.
3. Choose a color for the first month and display the bar.
4. Move over a few columns and display the second bar.
5. Do the same for the third month.
6. Print a title.

As with any long program, it was written in modules. (Recall that *module* is the buzzword for a piece of a program.) Most of the modules are similar to short programs. After you first decide what you want to do, you'll find that writing a program often consists of patching together modules you've already seen (perhaps in a different context). You make some small changes, write a control module, and you are done.

The IF-clause on line 130 has three separate tests. They check whether the data is out of range. When the IF clause is false, the data is out of range, so the PRINT statement tells you so and the program ends by going to line 430. Between lines 140 and 170 there is a timing loop so that you can read the statement telling you that your data is out of range. You should also notice that the GOTO command on line 170 doesn't send the //c to the END. The //c is sent to the TEXT:HOME combination before going to END. It is a good idea to leave a graphics program by first returning to TEXT. (You should make sure that the program doesn't erase the drawing before you get a chance to look at it. One good way to create a pause is to have the program ask you a question, using an INPUT command, before shifting back to TEXT.)



The only new ideas in this program are contained in the "graphing module." Although it is only eight lines long, it does many things. First, line 210 plots the axes for the graph. All the bars are four columns wide and are plotted in four passes through the loop. Each pass adds one column to each bar. Before the //c tries to plot a bar, it checks to see that the size of the bar is in the allowable range. Then the color is set and the //c starts plotting.

You might be wondering what would happen if you entered 2.2 units for the first month. Line 240 would tell the //c to plot a bar between rows 38 and 36.8. Of course there isn't any row 36.8 so the //c would truncate the .8 and it would plot the bar up to the 36th row.

You can change the colors used by this program to suit your mood. The COLOR command accepts values from 0 to 15. Line 210 sets the color of the border to 15, white. Bar colors are set in line 240 (2 = dark blue), line 260 (3 = violet), and line 280 (4 = dark green). To change the color used by the program, just change the number of the COLOR command on those lines.

Reading Bar Graph

Below is the data strip that contains the "Bar Graph" program. If you need additional help reading a data strip, refer to your reader instruction booklet. Your Cauzin Communications program also contains handy help screens to assist you.

After you've read in the strip, execute the program. This program is menu driven.

BAR GRAPH
by Gary Cornell & William Abikoff
The BASIC Apple //c

Softstrip
COMPUTER READABLE PRINT

Cauzin's Corner . . .

and now for something slightly different

This program works on any Apple II computer and runs fine under DOS 3.3 and ProDOS. It is written in Applesoft BASIC, so you can see how it works. To study the commands, type LOAD BAR.GRAPH and press the RETURN key. Then type LIST and press RETURN . You will see all the program lines scroll down the screen. Stop them by pressing CONTROL-S. Enter LIST 210 to see just one line, in this case line 210.

If you want a printout of the program, enter PR#1 and then enter LIST. Make sure your printer is turned on and ready to print. A printout helps you study the techniques used here.

The commands of the graphing module (lines 200-300) can easily be used in your own programs. As written, the program draws three bars. Change this by modifying line 220 and by adding a few lines to draw a fourth bar. Type the following lines, they will automatically fit in with the previous ones.

```
120 INPUT M1,M2,M3,M4
125 IFM4 < 40 THEN GOTO 140
220 FOR S = 10 TO 14
285 IFM4 < 1 GOTO 290
286 COLOR = 1: VLIN 39-M4,38 AT S + 18
```

Line 300 prints the label below the graph. Rewrite it to match both the number and the labels you need.

Bar Graph

```
10  REM A BAR GRAPH PROGRAM
11  REM
12  REM THE BASIC APPLE //C
13  REM
14  REM BY CORNELL AND ABIKOFF
15  REM
16  REM PAGE 175
17  REM
18  REM JOHN WILEY & SONS, INC.
19  REM
20  HOME
30  PRINT "THIS PROGRAM WILL DISP
     LAY A BAR GRAPH   FOR 3 MONT
     HS."
40  PRINT "YOU MUST MAKE UP THE U
     NITS YOURSELF."
50  PRINT "YOU CAN USE UP TO 39 U
     NITS IN ANY MONTH."
95  REM
96  REM
97  REM *************************
     *******
98  REM    DATA GATHERING MODULE
99  REM *************************
     *******
100  PRINT "WHAT ARE THE NUMBERS
     FOR THE 3 MONTHS?"
110  PRINT "ENTER THE 3 AMOUNTS S
     EPARATED BY COMMAS."
120  INPUT M1,M2,M3
130  IF M1 < 40 AND M2 < 40 AND M
     3 < 40 THEN  GOTO 200
140  PRINT "YOUR DATA IS OUT OF R
     ANGE."
150  FOR I = 1 TO 5000: NEXT I
170  GOTO 430
195  REM
196  REM
197  REM *************************
     *******
198  REM    GRAPHING MODULE
199  REM *************************
     *******
200  GR
210  COLOR= 15: VLIN 0,39 AT 0: HLIN
     0,39 AT 39
220  FOR S = 14 TO 17
230  IF M1 < 1 GOTO 250
240  COLOR= 2: VLIN 39 - M1,38 AT
     S
250  IF M2 < 1 GOTO 270
260  COLOR= 3: VLIN 39 - M2,38 AT
     S + 6
270  IF M3 < 1 GOTO 290
280  COLOR= 4: VLIN 39 - M3,38 AT
     S + 12
290  NEXT S
300  PRINT "MONTH =          1
      2     3"
395  REM
396  REM
397  REM *************************
     ********
398  REM    RUN AGAIN MODULE
399  REM *************************
     ********
400  PRINT "DO YOU WANT TO DRAW A
     NOTHER GRAPH?"
410  INPUT "(ENTER Y OR N) ";YN$
420  IF YN$ =  CHR$ (121) OR YN$ =
     "Y" THEN  TEXT : HOME : GOTO
     100
430  TEXT : HOME
440  END
```

The Lo-Res Graphics Editor

This section contains our "paintbrush" program, or lo-res graphics editor for any Apple II computer. This program is even longer than the bar graph program. Because it is so long, we had to carefully outline the program and build it in pieces. To give you an idea of how to write a long program, we'll take you through the steps we used. If you don't want to work through the details of such a long program but still want to use it, we suggest entering it and then saving it. Later you might want to see how it works and how it was written.

The first step in writing a program is to decide what you want it to do. We wanted to write a program that lets you move a paintbrush around the //c's grid and then use that brush to paint any block that you want. We wanted to permit the use of any of the sixteen colors that are available in lo-res graphics. Suppose the paintbrush is at a particular block. You should be able to move it up or down, left or right. The program should also be able to show the present position of the paintbrush as well as let you stop the program gracefully (without using CONTROL–C).

Then we started outlining the program. The painter—the person controlling the paintbrush—should be presented with a choice of things to do:

1. Find out where the paintbrush is.
2. Stop painting.
3. Move in any of four directions.
4. Decide to paint some color at the present position of the paintbrush.
5. After taking one of the above actions, the painter should be to take further actions.

This was a good first outline but it wasn't enough. For example, suppose the painter moved the brush off the screen and asked the //c to paint something off the sides. This would cause an error message; the program would automatically end and everything that had been painted would be lost. So before doing step 4 (or step 1) we should check that the new location is allowed. (In any complicated program, you try to *trap* or test for illegal commands before they bomb your program. We did this in the bar graph program.) It is rarely possible to trap every problem that might arise; usually programmers are satisfied to trap the most common problems.

Before we started writing the program we thought about how the //c would keep track of the position of the paintbrush. To do this we need two variables—one for the column and one for the row. Naturally, we decided to call them *COL* (for column) and *ROW*. Each time the painter asks to move left, the //c should subtract one from the variable named *COL*; each time the paintbrush moves down the //c should add one to the variable named *ROW*; and so on. The initial values for *COL* and *ROW* determine where the paintbrush starts. To start in the center of the grid, the count must start with *COL* = 20 and *ROW* = 20. (In computerese, we say that we *initialize* both of the variables to have value 20.)

Next we wrote an outline that showed the different modules that we would need. In long programs the first module always has the same form. It tells the operator what program is running, gives some directions, and asks whether to proceed or end the program. This is usually a good idea because you can run a program by mistake and it may take two hours to run. Of course you can always stop any program by using CONTROL–C, but sometimes you will destroy information on a disk by stopping a program in the middle.

In a program like the "paintbrush," where there are many choices, the second module should be a menu. Since moving the paintbrush is a simple command, we decided to include the moves inside the menu. The menu or "choice module" could also contain the exit from the program (the END statement). It is a good idea to keep instructions on the screen so that the operator is told what he or she can do. We thought that these instructions could be part of the "choice module" and just stay on the screen while other modules were being processed.

From our original outline, we knew we needed one module for actually painting the colors on the screen and another for locating the position of the paintbrush. It turned out that the "locate module" was the most interesting (and hardest) to write.

We decided to catch the painter's attention by having the position of the brush flash on the screen. (When a block flashes it changes color.) If the block had already been colored, we wanted to have the //c return the block to its original color after flashing. The //c will have to remember the original color of the block to do this. As soon as we knew exactly what we wanted the module to do, we could write a detailed job description for the module.

1. Check that the current *ROW* and *COL* are allowable positions (that they lie on the graphics screen).
2. Find out what color that position is currently painted.
3. Paint over that position with black.
4. Alternately paint the position black and white with enough time delay so that it can be seen.
5. Go back to the original color.

Step 1 is an IF–THEN. Steps 3, 4, and 5 use COLOR and PLOT commands with timing loops. Step 2 is the problem. So far we haven't introduced a command that can examine the screen and tell us the color that is painted at a particular location in the grid. We knew what we wanted to do but didn't know how to do it. Whenever this happens on an Apple or any other computer, we start going through a reference manual that lists all the available commands (for example, the reference section of an Applesoft manual). If we're lucky, we find exactly the command we need; if we're not, we must either program around the problem or rethink the whole module (or even the whole program). Fortunately, Applesoft has the command we needed.

The command we needed is SCRN($EXPRNM_1$, $EXPRNM_2$). $EXPRNM_1$ is the column and $EXPRNM_2$ is the row. SCRN(COL,ROW) gives the code for the color in the block at the current values of *COL* and *ROW*. Once we had this command we were able to translate our outline into another outline. The second outline was very close to the lines that we needed in the module. Because it is already close to being in Applesoft statements, we say that this outline is in *pseudo-code*. This buzzword means statements written somewhere between standard English and a programming language. We aren't sure where English stops and pseudo-code begins. Everyone seems to have their own definition of pseudo-code. An example in ours is:

```
if col < 0 or col > 39 then problems
if row < 0 or row > 39 then problems
c = scrn(col,row)
color = 0: plot color: delay loop
color = 15: plot color: delay loop
loop the previous two steps (for flashing)
color = c: plot color
go back to the menu
```

You should notice that at this stage we didn't say what we wanted to do if we had problems. The //c had to be sent to some module that does error trapping, but we hadn't written that module yet.

We next made a list of modules (see Table) with blocks of lines set aside for each module (we left plenty of extra lines in case any module turned out to be much longer than expected). In general, no module should be longer than about one page (25 lines). If a module gets too long, you should try to break it up into smaller modules.

TABLE

LIST OF MODULES

1. Direction module (lines 10-490)
2. Choice module (lines 500-990)
3. Locate module (lines 1000-1490)
4. Painting module (lines 1500-1990)
5. Error trapping module (lines 2000-)

After we laid out each module, we started writing the program. Then it was easy. For example, to write the *locate module* we had to use line numbers, problems became a GOTO statement, the delay loop became a timing loop and we wrote a loop around lines 4 and 5.

We'll go over each module separately.

The entry module for the program uses lines 10 to 150. The program starts and ends there, but it really doesn't do anything. It clears the screen and gives some instructions on how to use the program. When the //c sees the INPUT command it stops and waits for you to enter something at the keyboard. If you are ready to run the program, you enter *S*. Hitting any other key immediately ends the program.

After starting, the //c goes to the choice module in lines 500 to 650. First, this module puts the //c in graphics mode and prints directions in the text window. The paintbrush starts in the center of the grid. By following the directions, you can choose what the paintbrush should do. If it is to be moved, the move is made by changing the values of the *COL* and *ROW* variables. Then processing goes back to line 520. The directions are reprinted in the text window and the paintbrush awaits further orders. The reprinting is done almost instantaneously. As fast as you can choose a direction and hit *RETURN*, the paintbrush gets ready for its next action.

In line 600 you can choose to stop painting. (This INPUT statement also allows you to look at what you've painted.) If you choose to have the brush paint the block where it is located, the program goes to the paint module. If you want to find out where the paintbrush is, processing is sent to the locate module.

The paint module first checks to see whether the brush is positioned on the grid. Asking the //c to PLOT outside the grid bombs the program unless it traps the illegal command. If the brush is off the grid, the //c sends it back to the center without doing any painting (by going to the off-screen trapping module). If the block is on the grid, the program asks for a color and paints it using that color. It then goes back to ask for more instructions. When the program ends, the //c is still in graphics mode. This lets you keep your picture on the screen so people can look at it until you enter a TEXT:HOME combination.

We enjoyed writing this program because of the different tricks we had to use to make it work well. Even if you don't like writing very long programs, you can enter the paintbrush program and use it. We think it is fun to use.

CAUZIN'S CORNER . . .

and now for something slightly different

As written, the paintbrush program depends on INPUT statements. You could replace these with GET's for single key-press painting. Enter this line:

```
550 GET CH$
```

Now when you paint, you don't have to press the return key each time you move. However, this command is not as forgiving as INPUT. Once you press the key, your Apple paints.

Reading Paint Brush

Below is the data strip that contains the "Paint Brush" program. If you need additional help reading a data strip, refer to your reader instruction booklet. Your Cauzin Communications program also contains handy help screens to assist you.

After you've read in the strip, execute the program. This program is menu driven.

PAINT BRUSH
by Gary Cornell & William Abikoff
The BASIC Apple //c

Paint Brush

```
10  REM THE PAINTBRUSH
11  REM
12  REM THE BASIC APPLE //C
13  REM
14  REM BY CORNELL AND ABIKOFF
15  REM
16  REM PAGE 183
17  REM
18  REM JOHN WILEY & SONS, INC.
19  REM
20  TEXT : HOME
30  PRINT "THIS IS THE PAINT BRUS
    H PROGRAM"
40  PRINT "THE MOVES ARE: U=UP, D
    =DOWN"
50  PRINT "THE MOVES ARE: L=LEFT,
    R=RIGHT"
60  PRINT "USE P TO PAINT"
70  PRINT "USE THE W TO FIND OUT
    WHERE YOU ARE"
80  PRINT "USE THE E TO EXIT"
90  INPUT "ENTER S TO START ";S$
100  IF S$ = "S" THEN 500
110  PRINT : PRINT "YOU ENTERED S
    OMETHING STRANGE."
120  PRINT "TYPE RUN TO START AGA
    IN"
130  FOR PAUSE = 1 TO 2000: NEXT

140  TEXT : HOME
150  END
495  REM
496  REM
497  REM ************************
    ***
498  REM CHOICE MODULE
499  REM ************************
    ***
500  GR
510 ROW = 20:COL = 20
520  HOME
530  PRINT "U=UP, D=DOWN, L=LEFT,
    R=RIGHT"
540  PRINT "W=WHERE, P=PAINT, E=E
    XIT"
550  INPUT CH$
560  IF CH$ = "U" THEN ROW = ROW -
1: GOTO 520
570  IF CH$ = "D" THEN ROW = ROW +
1: GOTO 520
580  IF CH$ = "L" THEN COL = COL -
1: GOTO 520
590  IF CH$ = "R" THEN COL = COL +
1: GOTO 520
600  IF CH$ = "E" THEN 140
610  IF CH$ = "W" THEN 1000
620  IF CH$ = "P" THEN 1500
630  HOME : PRINT "TYPO. RETRY"
640  FOR P = 1 TO 1000: NEXT P
650  GOTO 520
995  REM
996  REM
997  REM ************************
    *****
998  REM LOCATE MODULE
999  REM ************************
    *****
1000  IF COL < 0 OR COL > 39 THEN
    2000
1010  IF ROW < 0 OR ROW > 39 THEN
    2000
1020 C =  SCRN( COL,ROW)
1030  FOR I = 1 TO 8
1040  COLOR= 0: PLOT COL,ROW
1050  FOR J = 1 TO 150: NEXT J
1060  COLOR= 15: PLOT COL,ROW
1070  FOR J = 1 TO 150: NEXT J
1080  NEXT I
1090  COLOR= C: PLOT COL,ROW
1100  GOTO 520
1495  REM
1496  REM
1497  REM ***********************
    ******
1498  REM PAINTING MODULE
1499  REM ***********************
    ******
1500  IF COL < 0 OR COL > 39 THEN
    2000
1510  IF ROW < 0 OR ROW > 39 THEN
    2000
1520  INPUT "WHAT COLOR DO YOU WA
    NT? ";C
1530  COLOR= C: PLOT COL,ROW
1540  GOTO 520
1995  REM
1996  REM
1997  REM ***********************
    ******
1998  REM OFF-SCREEN MODULE
1999  REM ***********************
    ******
2000  HOME : PRINT "YOU HAVE MOVE
    D OFF-SCREEN."
2010  PRINT "THE BRUSH HAS BEEN M
    OVED BACK"
2020  PRINT "TO THE CENTER. TRY A
    GAIN."
2030  FOR I = 1 TO 3000: NEXT I
2040  GOTO 510
```

Jumble

A jumble is a type of puzzle that is often found in newspapers. You get a bunch of letters and are supposed to rearrange them so they form a word. A jumble program needs a long list of words. It then calls a subroutine that jumbles a randomly chosen word from the list. After that, it asks the person running the program to guess the word. The //c can check whether a guess is correct with an IF-THEN that tests whether two strings are equal. The hard part of the program is the subroutine that jumbles the word. An outline is simple: randomly pull letters from the original word and reassemble them. Actually asking the //c to do this is a bit tricky. Each time a letter is chosen, it must be removed from the word that is being jumbled (otherwise it might be used again).

Thinking of how to accomplish a similar task in real life often helps to clarify the problem at hand. One analogous situation is removing a damaged link from a chain. First you must find the link you want to change. Remove the piece of the chain up to that link and put it aside. Then put the damaged link in a different place. Put the remaining piece of chain in a third place. Now you have pieces of chain in three different places. Finally take the first and third pieces of chain and reconnect them.

Each of these operations has a counterpart in Applesoft. The splitting is done by the MID$ command. The reassembling is done with the +. For example, to remove the eighth letter from a string (*ST$*) that is long enough, enter the command:

```
ST$ = MID$(ST$,1,7) + MID$(ST$,9)
```

This command concatenates (joins) the sub-string of *ST$*, (*sub-string* is the buzzword for a string that's part of a bigger string) consisting of the first through seventh characters, with the sub-string starting at the ninth character. If *ST$* started out with at least eight letters, it now has one less letter. Doing this repeatedly is the key step in the jumble program. Once the //c knows how many letters are in the string, the operation can be placed inside a loop.

Notice that we use a global variable, *TIM*, in the timing loop. This lets us get different pauses from one timing loop. You can change line 2080 to allow more or less time to guess the jumble. (You should also change the prompt on line 2070.) Other ways to improve this program are to make the DATA list longer and add a subroutine to keep score.

CAUZIN'S CORNER . . .

and now for something slightly different

Line 60 tries to get you better random numbers for the jumble program. When you begin to play the game, you enter a number and the computer uses that with the RND command to generate a new series of pseudo-random numbers. Don't cheat. If you enter the same number each time, you will get the same series of words.

Finally, a word game is only as good as the words it uses. Jumble uses computer-words stored in DATA statements—lines 5000 to 5030. Add lots of your favorite words in new lines. Type a line number between 5040 and 6000, next the word "DATA", and then your list of words separated by commas. For example:

```
5040DATA CAT,DOG,HOUSE,ELEPHANT,HAMBURGER
```

Use only capital letters, no spaces, and only about 200 letters in all. Add a good long list of new DATA statements to increase Jumble's vocabulary.

Reading Jumble

Below is the data strip that contains the "Jumble" program. If you need additional help reading a data strip, refer to your reader instruction booklet. Your Cauzin Communications program also contains handy help screens to assist you.

After you've read in the strip, execute the program. This program is menu driven.

JUMBLE
by Gary Cornell & William Abikoff
The BASIC Apple //c

Softstrip
COMPUTER READABLE PRINT

Jumble

```
10  REM A JUMBLE PROGRAM
11  REM
12  REM THE BASIC APPLE //C
13  REM
14  REM BY CORNELL AND ABIKOFF
15  REM
16  REM PAGE 326
17  REM
18  REM JOHN WILEY & SONS, INC.
19  REM
20  HOME
30  PRINT "DO YOU WANT TO TRY A J
    UMBLE?"
40  PRINT "TYPE ANY WHOLE NUMBER
    BUT 0 TO PLAY"
50  INPUT "THEN PRESS RETURN ";N
60 A =  RND ( - N): REM RESEED TH
    E RANDOM NUMBER GENERATOR
70  IF N = 0 THEN  GOTO 200
80  GOSUB 1000
90  GOSUB 1500
100  GOSUB 2000
110  PRINT "DO YOU WANT TO TRY AG
    AIN?"
120  INPUT "(ENTER Y OR N) ";YN$
130  IF YN$ = "Y" OR YN$ =  CHR$
    (121) THEN 80
200  END
995  REM
996  REM
997  REM ************************
    ******
998  REM PICK A RANDOM WORD FROM
    LIST
999  REM ************************
    ******
1000  RESTORE
1010 CT = 0
1020  READ A4$
1030  IF A4$ = "ENDWORD" THEN  GOTO
    1060
1040  LET CT = CT + 1
1050  GOTO 1020
1060  LET I3 =  INT (CT *  RND (1
    ) + 1)
1070  RESTORE
1080  FOR Q = 0 TO I3
1090  READ A4$
1100  NEXT Q
1110  RETURN
1495  REM
1496  REM
1497  REM ************************
    ******
1498  REM JUMBLE THE WORD
1499  REM ************************
    ******
1500 JMB$ = "":B4$ = A4$
1510 M4 =  LEN (B4$)
1520  IF M4 = 0 THEN 1590
1530  LET S4 =  INT (M4 *  RND (1
    )) + 1
1540 JMB$ = JMB$ +  MID$ (B4$,S4,
    1)
1550  IF S4 = 1 THEN B4$ =  MID$
    (B4$,2): GOTO 1510
1560  IF S4 = M4 THEN B4$ =  MID$
    (B4$,1,M4 - 1): GOTO 1510
1570 B4$ =  MID$ (B4$,1,S4 - 1) +
     MID$ (B4$,S4 + 1)
1580  GOTO 1510
1590  RETURN
1995  REM
1996  REM
1997  REM ************************
    *******
1998  REM DISPLAY THE WORD
1999  REM ************************
    *******
2000  HOME
2010  VTAB (12)
2020  PRINT "CAN YOU GUESS WHAT W
    ORD I'VE JUMBLED?"
2030  PRINT : PRINT
2040  PRINT  TAB( 20 -  LEN (A4$)
    / 2);JMB$
2050  PRINT : PRINT
2060  PRINT : PRINT
2070  PRINT "I'LL GIVE YOU ONLY 3
    0 SECONDS BEFORE    I'LL ERA
    SE IT.  (WAIT TILL THEN.)"
2080 TIM = 30: GOSUB 2500
2090  VTAB (12)
2100  INPUT "WHAT IS YOUR GUESS.
    (UPPER CASE ONLY!)      ";GU$
2110  IF GU$ <  > A4$ THEN  PRINT
    "SORRY THAT'S WRONG": PRINT
    : PRINT "THE CORRECT ANSWER
    IS ";A4$
2120  PRINT : PRINT
2130  IF GU$ = A4$ THEN  PRINT "C
    ONGRATULATIONS!"
2140 TIM = 5: GOSUB 2500
2150  RETURN
2497  REM
2498  REM
2499  REM *******TIMING LOOP*****
    *******
2500  FOR I = 1 TO TIM
2510  FOR P = 1 TO 700
2520  NEXT P
2530  NEXT I
2540  HOME
2550  RETURN
4998  REM
4999  REM
5000  DATA CLEAR,CONTINUE,DELETE,
    LIST,NEW,RUN,THEN,ERROR,RETU
    RN,RESUME,TRACE,CALL,MEMORY,
    DATA,DEFINE
5010  DATA FUNCTION,END,READ,WRIT
    E,STOP,NEXT,NEST,LET,RESTORE
    ,INPUT,PRINT,SPACE,CHARACTER
    ,STRING,COLOR
5020  DATA INVERSE,NORMAL,PLOT,PO
    SITION,CURSOR,SCALE,SORT,SCR
    EEN,DISPLAY,SPEED,BACKSPACE,
    BUZZER,PHYSICAL
5030  DATA LOGICAL,PROCESSING,EXE
    CUTION,SYSTEM,NUMBER,VARIABL
    E,LOOPING,GRAPHICS
6000  DATA ENDWORD
```

OTHER TITLES AVAILABLE IN STRIPWARE™

FOR APPLE

Art and Graphics on the Apple II/IIe
Essential Small Business Planning
Home Education, Vol. 1
Arcade Games, Vol. 1
Disk Doctor
Computer Puzzles
Balance Sheet

FOR IBM

PC Graphics
Balance Sheet
Classic Games
Eliza
Celestia
Personal Calendar
Home Applications

Softstrip
COMPUTER READABLE PRINT

- Create low resolution graphics.
- Make your own word jumbles.
- Design bar charts to help you analyze important data.

CAUZIN SYSTEMS, INC.
835 South Main Street
Waterbury, Connecticut 06706
(203) 573-0150

8501002

LIMITED WARRANTY

Cauzin StripWare™

835 South Main Street
Waterbury, Connecticut 06706

CAUZIN SYSTEMS, INC. (CAUZIN) warrants this package against defects in materials and workmanship for a period of 90 days from the date of original purchase.

If you discover a defect, CAUZIN will, at its option, either repair, replace, or refund the purchase price of this product at no charge to you, provided it is returned during the warranty period indicated above or any extended period(s) paid for by the user. Transportation charges should be prepaid, to the authorized dealer from whom you purchased it or to any other authorized CAUZIN dealer in the country.

You may obtain additional information from CAUZIN directly at the address printed on this certificate, or by calling (203) 573-0150. Please attach your name, address, telephone number, a description of the problem, and a copy of a bill of sale as proof of date of original purchase, to each product returned for warranty service.

This warranty applies only to products manufactured by or for CAUZIN, which can be identified by the CAUZIN trademark, trade name, or logo affixed to the product. CAUZIN does not warrant any product(s) that are not CAUZIN products. This warranty does not apply if the product has been damaged by accident, abuse, misuse or misapplication, has been modified without the written permission of CAUZIN, or if any CAUZIN serial number has been removed, altered, or defaced.

<u>ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO 90 DAYS FROM THE ORIGINAL DATE OF PURCHASE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTION ON THE FACE HEREOF.</u>

<u>THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED. NO CAUZIN DEALER, AGENT, OR EMPLOYEE IS AUTHORIZED TO MAKE ANY MODIFICATION, OR ADDITION TO THIS WARRANTY.</u>

<u>CAUZIN IS NOT RESPONSIBLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY BREACH OF WARRANTY, OR UNDER ANY OTHER LEGAL THEORY, INCLUDING LOST PROFITS, DOWNTIME, GOODWILL, DAMAGE TO, OR REPLACEMENT OF EQUIPMENT AND PROPERTY, AND ANY COST OF RECOVERING, REPROGRAMMING OR REPRODUCING ANY PROGRAM OR DATA STORED IN OR USED WITH CAUZIN PRODUCTS.</u>

Somes states do not allow the exclusion or limitation of incidental or consequential damages or limitations on how long an implied warranty lasts, so the above limitations or exclusion may not apply to you.

This warranty shall not be applicable to the extent that any provision of this warranty is prohibited by an federal, state or municipal law which cannot be preempted. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.